2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing

Energy Efficient Client-centric Shaping of Multi-flow TCP Traffic

Ahmad Nazir Raja, Zhihua Jin, Matti Siekkinen
Aalto University
School of Science and Technology
ahmadnazir@gmail.com, {zhihua.jin,matti.siekkinen}@tkk.fi

***Abstract*—Energy consumption is a concern with mobile devices nowadays. Network interfaces are among the most power hungry components in these devices. In this paper, we describe the design and implementation of a client-centric protocol for energy efficiency. Inspired by earlier work, our protocol works by exploiting the TCP flow-control mechanism to shape incoming traffic into bursts in order to utilize the unused bandwidth between the server and the client. Our solution works with multiple simultaneous connections and it is extensible to different scheduling policies. Furthermore, the purely client-centric nature of the protocol enables easier deployment of the solution. The protocol is application independent and it is targeted for bulky TCP transfers. The solution can be used on Linux based systems and is kept portable as it can be deployed with different wireless devices without major modifications. We have tested our solution with real web-servers and our results show that the protocol can achieve sleep time up to 80% and 45% of total duration of one and four active simultaneous connections, respectively.**

I. Introduction

Energy consumption is a great challenge for the battery powered mobile devices nowadays. Wireless network interfaces are among of the most power hungry components of these devices. To alleviate this problem, 802.11 includes *Power Save Mode* (PSM) which is a mechanism for dynamically switching the WLAN interface to a low power, sleep state when idle and back up again when new data is arriving or being sent.

While PSM works well for bulk traffic or data sent in bursts, as reported in [5], [8], it is inefficient when applied to a constant bit rate type of traffic common for streaming or peer-to-peer client upload traffic, for instance. Therefore, solutions to shape this kind of traffic into bursts have been proposed, such as the PSM Throttling mechanism in [10]. Inspired by this solution, we designed and implemented a similar kind of protocol. The advantages of our solution are the following: it is client-centric, i.e. requires no support from infrastructure or other end point, the architecture is hardware independent, it works with multiple connections which is necessary to be used with e.g. peer-to-peer applications and it is extensible so that different scheduling policies can be plugged in. The implementation is portable across different Linux platforms.

Several other related solutions have been proposed in the past. Some of these discard PSM and propose a different protocol such as in [8], [2], some suggest co-operative mechanisms between end points [7], [11], and some propose server-side [5], [4] or proxy-based solutions [9]. There are also solutions which trade off performance (delay, throughput) with energy savings such as in [12], [13]. In contrast, the starting point for our work was to come up with a deployable solution which would not require server side support or additional proxies, works with off-the-shelf 802.11 mobile devices, and does not hurt the application-level performance.

We also report evaluation results of our solution from tests in an infrastructure WLAN. Note that it should also work fine in an ad-hoc configuration because of the client-centric nature of the protocol. We focus only on TCP traffic which is the most common transport layer protocol used for both web browsing and bulk data transfers (i.e. file downloads as well as multimedia streaming). Our technique is based on exploiting TCP's flow control mechanism and, hence, does not support UDP traffic. Furthermore, the protocol is not suitable for very interactive applications or applications with sporadic traffic patterns such as gaming.

We discuss our client-centric solution in Section II. Section III deals with the design of our solution by discussing the core components and implementation related details. Section IV contains the experiments and their outcomes along with our analysis of the results and possible extensions to our work. We conclude in section VI.

II. Client-centric Traffic Shaping using PSM Throttling

In this paper we present a novel technique for achieving energy efficiency for bulky TCP transfers in a client-centric manner. The design is kept hardware independent and works with multiple connections, which we have demonstrated through performance evaluation of a real prototype. As our solution is client-centric, it does not require additional infrastructure support like the 802.11 PSM does. The design is inspired by the PSM Throttling protocol proposed by *Tan et al.* in [10] that detects unused bandwidth between the server and the client and uses it to increase predictability of packet arrival and shape traffic into bursts. High predictability of incoming traffic can be used to switch the state of the WNI for energy efficient communication. The technique effectively utilizes available Internet bandwidth without degrading the application's performance as perceived by the user. We have borrowed many mechanisms from the protocol but have considered the addition of the following features while designing our solution:

- Use of timers to transition between states, if required

978-0-7695-4331-4/10 $26.00 © 2010 IEEE
DOI 10.1109/GreenCom-CPSCom.2010.135

- Use of TCP Keep Alive packets to avoid deadlocks
- Keeping the TCP functionality intact by implementing our solution as a separate layer
- Dynamic calculation of advertised window sizes for maintaining throughput
- Adding functionality for multiple connections
- Hardware independence of the solution

Our solution works by exploiting the TCP Flow Control Mechanism which is a windowing mechanism used to control the flow-rate between the sender and the receiver. According to the mechanism, the receiver controls the amount of data that can be received with the help of *Receiver Window Size*. A window size with a zero value means that the buffer is full and the client can not accept more data. Such a packet is referred to as a choke ACK.

Similar to PSM-Throttling, our solution consists of two parts: *Bandwidth Throttle Detection* and *Traffic Burst Generation*. The first part as described in [10] detects unused end-to-end bandwidth restricted by the server, in order to determine whether or not to proceed with the second part. First, the client measures the flow rate *r* as a baseline. Then, the client pauses the server data transfer by sending a choke ACK. After two RTTs, the client resumes the server data transfer by sending an unchoke ACK with restored window size. The client expects 2RTT*r bytes of data buffered at the server to arrive in a burst, utilizing the full end-to-end bandwidth r'. If $r' > r$, there is unused bandwidth that can be used for traffic shaping.

Traffic Burst Generation is the actual mode of operation of our solution. The incoming packets are shaped into bursts similar to how bandwidth is detected during throttle detection. The client sends ACKs with specific window sizes and then waits for the amount of data to arrive. On the arrival of the first packet, the client acknowledges it by sending a choke ACK preventing the server to send in more data. The client keeps on receiving data for 1 RTT after having sent the choke ACK. It counts the number of bytes received and sends an open ACK once all the expected data has arrived. Let the time taken to receive the burst be T_{recv} and the interval between be T_{idle}, the total burst time T_{burst} is equal to the sum T_{burst} = (T_{recv} + T_{idle}), then it makes sense to shift the WNI to the sleep state for time period T_{idle} as long as it is non-trivial (i.e. $T_{idle} \geq$ RTT) otherwise the overall savings might turn out to be insignificant. We have noticed in our experiments that the server doesn't always send data with size equal to the receiver window size. In such cases, the client doesn't totally rely on counting the incoming bytes before transitioning back to the sleep state but does so after a predetermined amount of time i.e. the RTT since the last window advertisement was sent.

It is very important to set the value of the receiver window size small enough that that all the data can arrive within an RTT. If the number of bytes, as specified in the window size, takes more than 1 RTT to reach the client, then the algorithm will fail as the subsequent choke ACK will prevent the server from sending more data and the client will keep on waiting for the data to arrive. Therefore, an appropriate sized window should be set to avoid any deadlock. On the other hand, a very small sized window will also lead to a lot of overhead and the protocol might consume more energy than normal. Hence, the the window sizes need to be advertised dynamically to maintain the throughput.

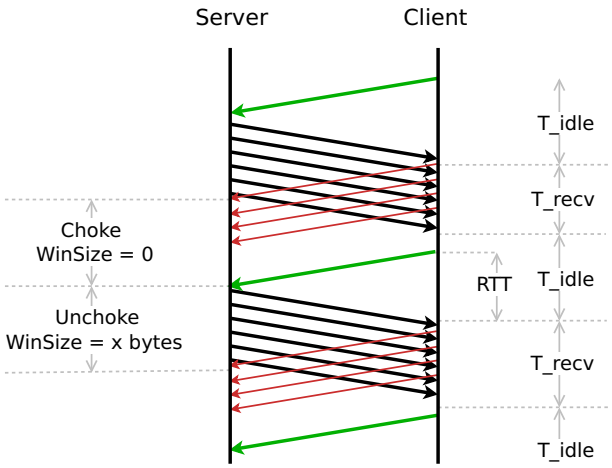

Fig. 1. Generating bursts using the PSM-T protocol

III. Design and Implementation

Following are the requirements we set for the solution:

1) *Portable through Hardware In-dependency:* Even though, control of the wireless device might need some sort of modification of the device driver, the implementation of the energy saving mechanism should be independent of the hardware and it should be possible to implement the protocol with other wireless devices.
2) *Easy to Deploy:* The end result should require no (or little) modifications to existing functionality of the TCP/IP protocol stack. In addition, no support from infrastructure (APs, middle-boxes) or other end point should be necessary.
3) *Low Computation Overhead:* Since, the overall energy saved depends on how often the WNI is transitioned to the sleep state, it is important to make these decisions quickly. In addition, to save energy, the computational overhead of the solution should be as small as possible.

Consequently, the design of our solution consists of three components: *Traffic Shaping Protocol*, *Scheduler* and *WNI control*. The design relies on the Linux kernel which makes it portable from a Linux system to another but not as such to Symbian, for instance. We implemented these three design components as a single kernel module. We explain the roles and detailed design of the three components individually below.

A. Traffic Shaping

The function of the traffic shaping component is to, namely, shape the incoming traffic into bursts so that the underlying

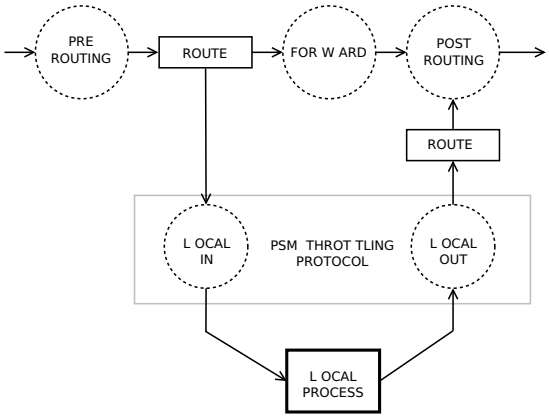

Fig. 2. Using appropriate Netfilter hooks

device can be put to sleep in between receiving these bursts. The bursts are generated as we explained in Section II, and this component needs also to ensure that throughput is not significantly affected. The shaping mechanism operates between the network and data-link layer of the TCP/IP protocol stack so that it has access to the TCP and IP information for both outgoing and incoming packets.

We have chosen to use the hooks provided by the *Netfilter Framework* (http://www.netfilter.org/) within the Linux Kernel for packet interception and modification. The framework provides five positions or hooks where functions can be registered: *Pre-routing*, *Post-routing*, *Forward*, *Local In* and *Local Out*. Figure 2 represents the five positions that Netfilter framework exposes. We use the *Local In* and the *Local Out* Hooks for registering our functions to implement the traffic shaping protocol. Intuitively, the function registered at the *Local Out* hook is responsible for out going packets and performs tasks such as modification of the receiver window size in the TCP header, if required. Similarly, the function registered at the *Local In* hook is responsible for all the incoming packets and is responsible for a number of tasks depending on the state of the connection. During throttle detection, the function determines the payload of the TCP packets and calculates the flow rates. During traffic burst generation, the function keeps track of the amount of data arrived in a burst and informs the scheduler accordingly. In addition, both the functions need to identify the specific TCP connection that the packet belongs to, which is done by checking the TCP source port that acts as the connection identifier.

Apart from normal operation, there can be cases where the protocol gets stuck in a deadlock. We rely on TCP Keep Alive packets in such cases, which help in resuming the connection and also in determining whether the connection has terminated or not. The TCP Keep Alive packet is simply a TCP ACK with zero payload.

For each connection, the shaper keeps track of states related to operational modes i.e. TCP Handshake, Throttle Detection, and Shaping in order to know when to perform shaping and when not.

An important responsibility of the shaper component is to determine the right size for receiver window advertisements. A too large window will cause the protocol to continue waiting for the complete burst (until the timer expires) which will keep the WNIC awake and increase the energy consumption. On the other hand, a too small window will reduce the throughput since a smaller number of bytes will arrive within a fixed period of time. Hence, the shaper adjusts the advertised window size as follows: Initially, the window size is calculated by taking the product of the reference throughput (detected at the beginning of the connection) and the RTT. This window size is advertised and the resulting throughput is constantly monitored. We maintain two thresholds for throughput in our implementation. The upper threshold is necessary because we noticed that in practice, depending on the TCP implementation, if the number of bytes advertised in the receiver window is not a multiple of MSS, the server might not send a full receiver advertised window worth of data leading to a slightly lower than expected throughput. Hence, we need to introduce a kind of safety interval into the protocol. If the monitored throughput is below the lower threshold, the window size is increased by the *Maximum Segment Size* (MSS) which we found to be 1452 bytes in our experiments. The throughput thresholds along with other protocol tuning parameters can be configured and are mentioned in Table I.

B. Scheduler

The scheduler is called when there is a change in the state of the TCP connection i.e. idle or busy. The scheduler is responsible for determining when to transition to the sleep state and when to wake up based on the states on all active connections. Once such an event should occur, the scheduler invokes the WNI Control which takes care of modifying the wireless device's physical state based on the command received.

To calculate an accurate value of the time instance that the network interface should be woken up, it is important to know some of the hardware specific parameters e.g. the time taken to transition from the sleep to wake state. Also, the network conditions should be keep in mind to predict arrival time of bursts. We use the RTT Variance value and a factor of it, *Aggression Factor*, to determine the arrival times. Figure 3 illustrates the impact of these parameters. Once we know the values of these parameters, we can compute the wake up time as follows:

$$T_{wakeup} = T_{ack} + RTT - (RTTVar * AF) - TT_{wake}$$

where,
T_{wakeup} = Wakeup time of the WNIC
T_{ack} = Time of Sending the Acknowledgment
AF = Aggression Factor
TT_{wake} = WNIC Transition time from sleep to wake state

A number of protocol tuning parameters are mentioned in Table I. Furthermore, hardware specific parameters such as

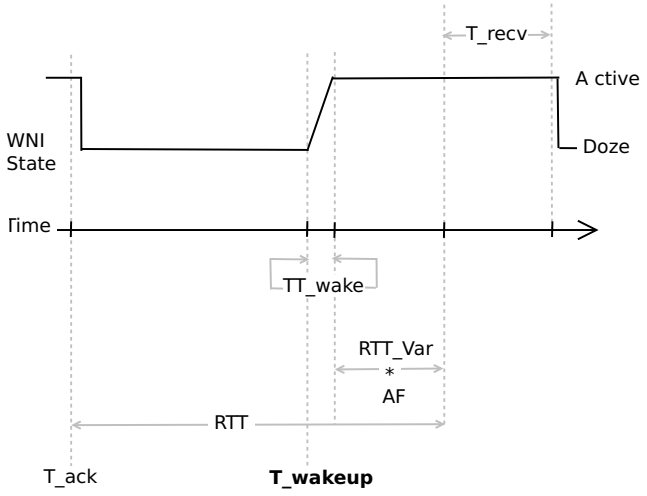

Fig. 3. Calculating the next Wakeup time of WNI

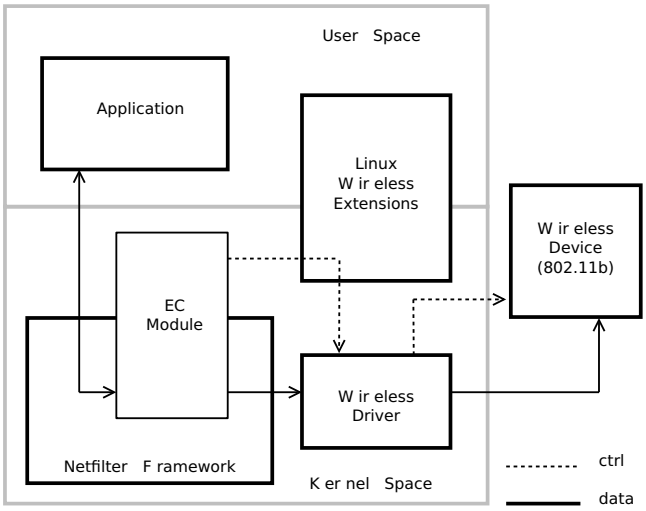

Fig. 4. The Architecture

transition times can also be tuned to cater for different hardware devices. According to [10], mode switching overhead i.e. between Power Save Mode (PSM) and Constant Awake Mode (CAM), is 4ms for Atheros chipset Wireless Network Interface (WNI). We also assume 4ms for the sleep to wake transition time in our experiments. On the contrary, the transition time from the wake to sleep state is negligible [3]. These parameters can be tuned according to the specifications of the underlying wireless device and hence device hardware independence is achieved.

C. WNI Control

The WNI Control is responsible for transitioning the WNI state to sleep or awake based on the scheduler's decisions. Note that in cases where the shaping is not considered a good choice, the device can just rely on standard PSM and this component (neither the scheduler) will never be invoked. We have used the Linux Wireless Extensions (LWE) to implement the mode switching feature. LWE use a set of user-space tools to configure the wireless device with the help of IOCTL calls. We reverse engineered the specific IOCTL call for mode switching and implemented the same functionality from the kernel domain.

Since transitioning to the sleep state is a hardware dependent feature, we just provide stub functions for transitioning the WNI to sleep and awake states. These functions can be implemented by the user who intends to use the protocol with the hardware device of choice.

D. Architecture

Figure 4 displays the architecture of the overall solution. The part overlapping with the Netfilter framework is responsible for interception and modification of TCP packets. The other part is responsible for scheduling and changing the states and operational modes of the wireless device.

The solid lines represent the data flow where the data is generated and received by the user-space application which can be a browser, media player, FTP client etc. The data is passed to the kernel so that it is processed by the TCP/IP stack on its way out or in. It also passes through the Netfilter framework and through the hooks where we have placed our code for traffic shaping. The dashed lines show how the operational mode of the wireless device is controlled. The figure only shows the use of Linux wireless extensions to transition between CAM and PSM and does not describe WNI control which is a hardware dependent function.

IV. Experimentation and Results

We experimented with the implementation of our protocol using a laptop (Intel Core Duo 1.83GHz T5600, 1 GB RAM) running Ubuntu 9.10 (Karmic Koala). The network controller used was Intel Corporation PRO/Wireless 3945ABG using iwl3945 wireless driver. The access point we used for our tests was Linksys Compact Wireless-G Broadband Router (Model: wrt54GC v2.0). However, the implementation was also tested on other Linux based machines, such as Nokia N900. Parameter values used for our experiments are show in Table I.

Our parameter for energy savings is idle time of the connection i.e. the time for which the connections are predicted to be in the idle state and for which the WNI can be put to sleep. We believe that this method of determining energy savings leads us to accurate results as the relationship between actual energy saved and the idle time is linear. The energy consumption while the device is active depends very little on the operating state [6] and for that reason a constant value is usually used for calculating the energy consumption. For Enterasys Networks RoamAbout interfaces, energy consumption while the device is in active state is about 750mW and during idle is 50mW [8]. Also, the energy consumption during sleep to wake transition

TABLE I
PROTOCOL TUNING PARAMETERS

Phase	Parameter	Purpose	Impact	Value range	Value Used
Bandwidth Throttle Detection	Playout Buffer Wait	Time period to wait at the start of the connection in order to skip the initial high rate data transfer in cases such as video streaming	Too large value wastes energy and time whereas too small value results in inaccurate measurement of the flow-rate	>0 secs	5 secs as used in [1], [10]
	Flow-rate Calculation Period	Time period to measure the reference flow-rate	Too large value wastes energy and time whereas too small value results in inaccurate measurement of the flow-rate	>1 RTT	5 secs
	Bandwidth Ratio	Least expected ratio between the end-to-end bandwidth and the reference flow-rate against. The protocol enters the *Traffic Burst Generation* phase if the ratio is greater than this value.	Smaller value allows the protocol to engage even when the unused bandwidth is not significant.	>1	2 as used in [10]
	Choke Factor	Time period in multiples of RTT to choke the server	A small value may result in inaccurate measurement of the available end-to-end bandwidth. A large value might also lead to inaccurate measurements since we don't know how the server will send the buffered data accumulated over a larger time period.	>2	3
Burst Generation	PSM-T Throughput Thresholds	Upper and lower bounds of acceptable throughput	Too small upper bound degrades the throughput whereas too large lower bound saves less energy	[0 - 100, 0 - 100] in theory	[95,97]
	Aggression Factor	The aggression factor is the value which determines how aggressive is the prediction scheme for incoming packets	Large value reduce the risk of missing the incoming packet, but leave less margin to save more energy	0 - 1	0.5

is almost the same as that in active state whereas energy consumption is negligible while transitioning from wake to sleep state [8].

A. Testing with Single Connections

We started with testing the protocol with a single Youtube connection where the RTT and Variance values for the connection were 46 msecs and 22 msecs respectively. Figure 5 is a graph acquired by plotting the idle time detected by the PSM-T protocol with the time lapsed since the connection entered the PSMT state. The connection time lapsed does not account for the initial 10 secs for the play-out buffer and flow-rate calculation. It can be observed that the relationship is linear which means that the protocol is capable of providing a constant level of energy savings.

Figure 6 plots the percentage of savings i.e. (Idle Time/Time Lapsed) * 100, with the time lapsed. It can be observed that there is a very steep increase in the accumulated savings because the actual savings start after some initial delay. This delay be explained by the fact that the protocol might take some initial time to stabilize and doesn't save energy if the throughput doesn't fall in the required thresholds. After the initial stabilization period, the protocol maintains the required throughput by dynamically adjusting the advertised window sizes. We can see from the figure that for a video longer than 100s, it is possible for the network interface to sleep more than 70% of the time. The maximum sleep time achievable in this experiment seems to be roughly 75%.

Figures 7 and 8 plot the results for a different connection

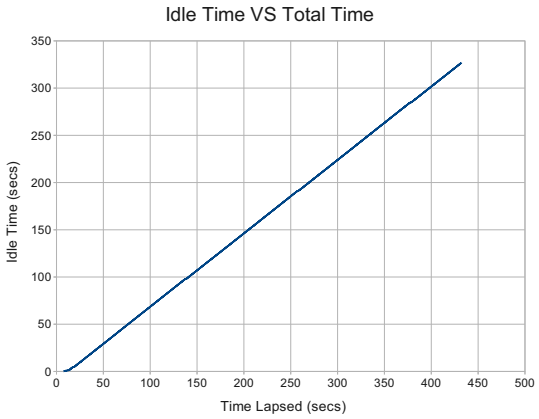

Fig. 5. Stability of a Single Youtube stream using PSM-T protocol

which experienced some RTT or bandwidth fluctuations at certain points of time. We observe that the protocol stopped accumulating idle time for about 50 seconds at approximately 240 secs and 420 secs. This can be explained from figure 9 which shows that the throughput dropped below a certain level (95% of the reference throughput) at those time instants. At those moments, the protocol tried to achieve the original throughput by stopping the scheduling and increasing the window advertisements. As soon as the required throughput was achieved, the protocol started accumulating idle time again. These fluctuations caused the accumulated sleep time to be only 55% of whole duration. By using more lenient

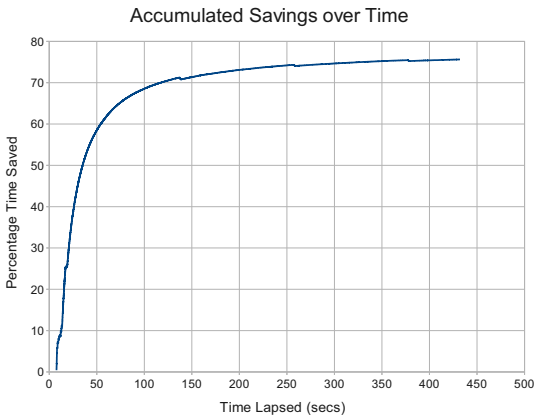

Fig. 6. Accumulated Savings of the PSM-T protocol over time

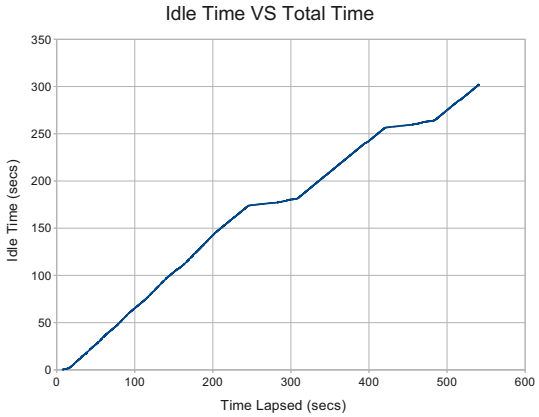

Fig. 7. Fluctuations in a single Youtube stream using PSM-T protocol

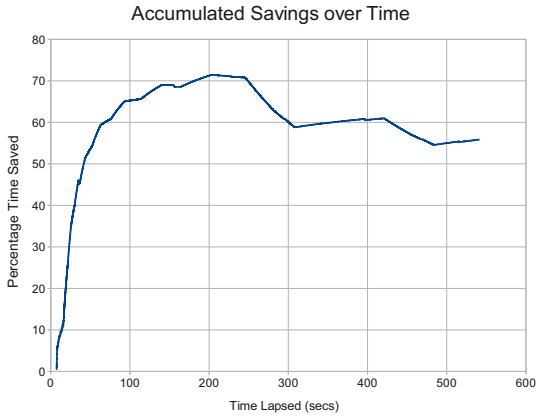

Fig. 8. Accumulated Savings with fluctuations from the Server

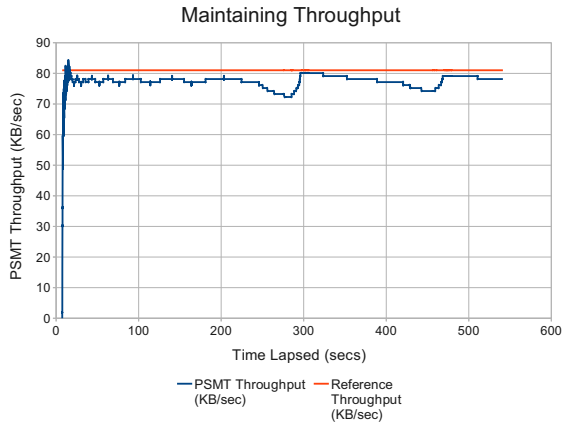

Fig. 9. Throughput Maintenance with fluctuations from the Server

throughput thresholds, the energy savings can be increased at the cost of throughput.

B. Testing with Four Active Connections

In this section we discuss the results of downloading four simultaneous Youtube streams which are established with 20 secs interval. This means that once a connection is setup, it takes about 30 secs for the next connection to enter shaping state. Since the graphs show effects of connections in shaping state only, we expect to see some fluctuation at about 30 secs intervals in the overall energy savings. Table II displays the RTT and variance values for the four connections.

	RTT (msecs)	Var (msecs)
Connection#1	45	23
Connection#2	44	24
Connection#3	41	13
Connection#4	45	22

TABLE II
RTT AND VARIANCE VALUES FOR THE CONNECTIONS

Figure 10 presents the stability as well as the accumulated savings for each connection. The stability of the protocol can be seen in the form of a linear relationship between the time saved (or idle time) and the total time. The accumulated savings graph for each connection show how quickly the PSM-T protocol reaches its maximum performance for each connection. It can be observed that apart from some fluctuation at around 230 secs in connection#1, all other connections are fairly stable. This fluctuation is expected to be seen in the overall sleep time. We see that due to the fluctuations, connection#1 achieves slightly less accumulated sleep time (70%) compared to connection#2, connection#3 and connection#4 (over 80%).

Figure 11 shows the combined effect of the active connections on energy savings. Note that these values represent the actual sleep time for the network interface. It is evident that

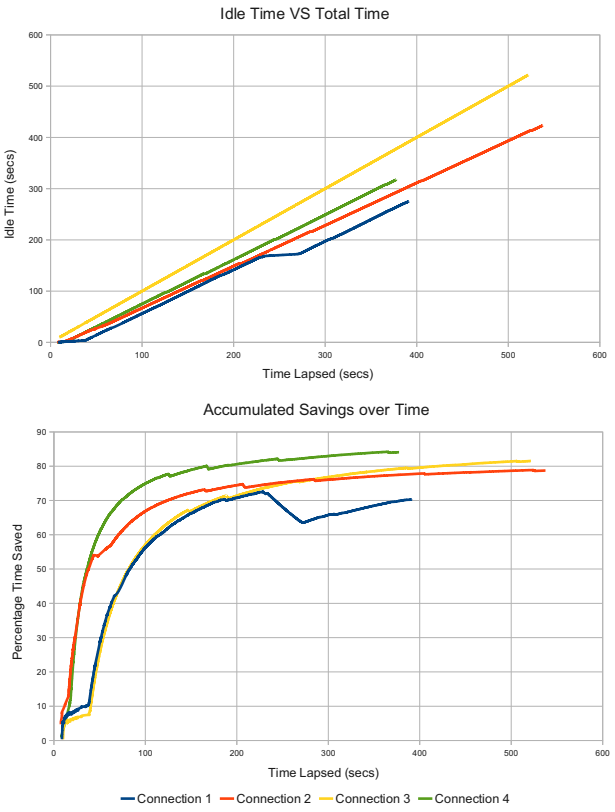

Fig. 10. Results for the four connections using PSM-T protocol

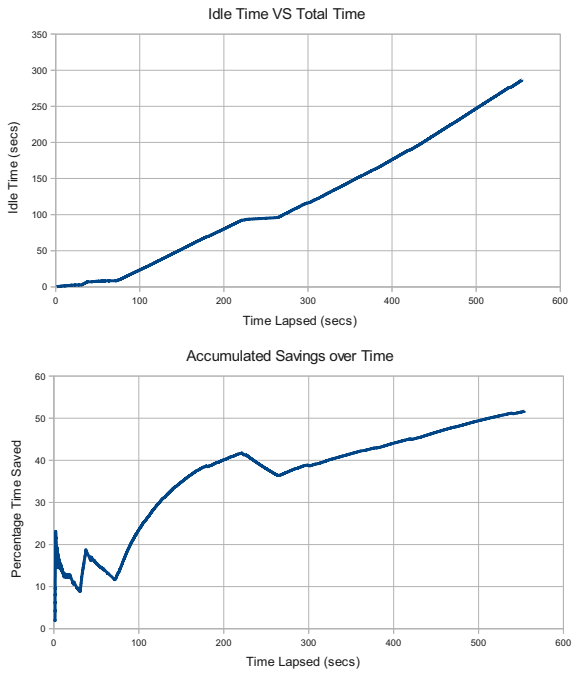

Fig. 11. Accumulative effect on savings - Emulating the WNI with four active connections using PSM-T protocol

the relationship between the idle time and time lapsed is not as linear nor as steep as that of individual connections because the bursts of different connections are not all overlapping. It seems that the relationship is steeper after 390 secs which is due to the termination of connection#1 and subsequently other connections. Since each connection is setup after 20 seconds from the previous one, it takes about 30 secs for next connection to enter the shaping state. The graph with accumulated savings plotted over time shows that the combined energy savings stabilize after 90 secs which is the time when all four connections enter the shaping state. There is a decrease in accumulated savings at about 220 secs which is the effect of connection#1 facing fluctuations. We see that it is possible for the interface to sleep 44% of the 390 secs (termination time of connection#1).

V. Discussion and Future Work

The shaping mechanism for energy savings relies on transitioning the WNI to the sleep state between two bursts. There is some overhead associated with each transitioning. Hence, the fewer transitions there are (i.e. the larger the interval) the greater are the savings. Currently, we use straightforwardly the RTT of the connection as the period during which a burst is generated and received.

However, we could increase this period by adding a constant delay, *delta* to the RTT, at the client end. In principle, as long as the advertised window size is equal to the bandwidth-delay product, the protocol should function correctly. In our case, the bandwidth is not the path capacity between the server and the client, but the reference bandwidth (FR_{normal}) calculated at the start of the connection. Thus, it should be possible to increase the efficiency of the protocol by increasing both the period and the advertised window size (WS) so that the following equality holds:

$$FR_{normal} * (RTT + delta) = WS$$

However, this artificial increase might have some affect on the throughput in real life scenarios. One of the possibilities include increasing the RTT to more than a certain limit in which case the server might start sending TCP Keep Alive packets. In addition, the receiving TCP's buffer size (can be relatively small in a mobile device) might limit the advertised window size. Studying the limitations of this approach is part of our future work.

Another avenue for future work is to design smarter scheduling policies. Currently, the scheduler we use is dumb: It is invoked by the traffic shaper component when the state of the connection is changed and if it notices that all the connections are in the idle state, it in turn tells WNI Control to put the interface to sleep. Since, the scheduler is aware of the states of all the connections and their attributes, it can be designed to intelligently adjust the period of generating and receiving a burst in order to synchronize the burst reception among all

active connections. It is non-trivial to design such a scheduler that also takes into account the fact that in real world transfers do not behave in a static manner.

VI. Conclusion

We present in this paper the design and implementation of a client centric and application independent protocol to save energy in bulky TCP downloads without affecting the average throughput of the connection. Our solution is portable among different Linux-based systems. It also works with multiple simultaneous connections and supports different scheduling policies. Furthermore, our implementation of the protocol complements standard TCP functionality and does not require modifications to it. We have tested our solution with real web-servers and have found that the protocol yields good results in most cases. However, for multiple simultaneous connections, a smarter scheduling would be beneficial, which is part of our future work.

Acknowledgment

This work was supported by TEKES as part of the Future Internet program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

The authors would like to thank Mohammad Ashraful Hoque for sharing his interesting ideas and solutions during the initial stages of the research.

References

[1] How to change the proxy and buffer settings in windows media player. http://support.microsoft.com/kb/257535.

[2] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189, New York, NY, USA, 2003. ACM.

[3] G. Anastasi, M. Conti, E. Gregori, and A. Passarella. Saving energy in wi-fi hotspots through 802.11 psm: an analytical model. In *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation, ESSLLI-2000*, pages 24–26, 2004.

[4] Surendar Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *In MMCN*, pages 85–99, 2002.

[5] Surendar Chandra and Amin Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *ATEC '02: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, pages 329–342, Berkeley, CA, USA, 2002. USENIX Association.

[6] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *In IEEE Infocom*, pages 1548–1557, 2001.

[7] Kyu-Han Kim, Yujie Zhu, Raghupathy Sivakumar, and Hung-Yun Hsieh. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. *Wirel. Netw.*, 11(4):363–382, 2005.

[8] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 119–130, New York, NY, USA, 2002. ACM.

[9] Prashant Shenoy, Peter Radkovdepartment, and Computer Science. Proxy-assisted power-friendly streaming to mobile devices. In *In MMCN*, pages 177–191, 2003.

[10] Enhua Tan, Lei Guo, Songqing Chen, and Xiaodong Zhang. Psm-throttling: Minimizing energy consumption for bulk data communications in wlans. *Network Protocols, IEEE International Conference on*, 0:123–132, 2007.

[11] V. Tsaoussidis and C. Zhang. Tcp-real: receiver-oriented congestion control. *Comput. Netw.*, 40(4):477–497, 2002.

[12] Haijin Yan, Rupa Krishnan, Scott A. Watterson, David K. Lowenthal, Kang Li, and Larry L. Peterson. Client-centered energy and delay analysis for tcp downloads. In *In IEEE International Workshop on Quality of Service*, 2004.

[13] Haijin Yan, Scott A. Watterson, David K. Lowenthal, Kang Li, Rupa Krishnan, and Larry L. Peterson. Client-centered, energy-efficient wireless communication on ieee 802.11b networks. *IEEE Transactions on Mobile Computing*, 5(11):1575–1590, 2006.